Mapping Windows ACLs into POSIX ACLs

By Jeremy Allison

Development Team

email: jra@samba.org

email: jeremy@valinux.com

Why attempt to map Windows to POSIX ACLs in Samba ?

- Windows administrators are used to simple ACL controls.
- The Samba mapping of UNIX user/group/world triple is not considered enough granularity for Windows permissions.
- Competing SMB implementations impelment Windows ACLs.
  - It becomes a 'checkbox' feature, no matter how used.
- Fits with Samba philosophy of allowing OS to control access, less user-space security policy.

POSIX ACLs - the non-standard standard.

- Not an official POSIX standard.
- Draft standard 1003.1e revision 17 is the API Samba standardized on.
- Differences in vendor implementations of this API mean Samba needs a interface layer to map to underlying OS.
  - Linux - the UNIX defragmentation tool - uses 1003.1e draft 17 as the API - so identity mapping used on Linux.
- Implementation of ACL support in Samba has increased pressure on ACL standardization.

POSIX ACLs

- Are extension of UNIX u/g/w permissions.
- Designed for simplicity. Allow additional users and groups to have access specified to a file or directory.
- Do not extend UNIX permission model with extra modes of access (rwx only).
- Two extra features added, inheritance (for directories) and masks.
  - Inheritance applied to files and directories alike.
  - Masks override group and additional permissions.

Examining a POSIX file ACL

- Sample POSIX file ACL :

```
# file: testfile                <--- file name
# owner: jeremy                 <--- file owner
# group: users                  <-- POSIX group owner
user::rwx                       <-- perms. for file owner (standard 'user')
user:tpot:r-x                  <-- perms. for extra user 'tpot'
group::r--                      <-- perms. group owner (standard 'group')
group:pcguest:r--               <-- perms. for extra group 'pcguest'
mask:rwx                        <-- mask 'ANDed' with groups and extra users
other:---                       <-- perms. for any other user (standard 'world')
```

Examining a POSIX directory ACL.

- Sample POSIX directory ACL :

```
# file: testdir/                <-- File name
# owner: jeremy                 <-- File owner
# group: jeremy                 <-- POSIX group owner
user::rwx                       <-- perms. for directory owner (standard 'user')
group::rwx                      <-- perms. for group owner (standard 'group')
mask:rwx                        <-- mask applied (ANDed) to group perms.
other:r-x                       <-- perms. for all other access (standard 'other')
default:user::rwx               <-- Inherited owner perms.
default:user:tpot:rwx           <-- Inherited extra perms for user tpot
default:group::r-x              <-- Inherited group perms.
default:mask:rwx                <-- Inherited default mask
default:other:---               <-- Inherited other perms.
```

POSIX ACL rules

- There are some special rules applied.
  - As all POSIX creation calls specify a default mode_t (created permissions) argument, then the most restictive set of inherited and requested permissions is used on creation of a filesystem object.
  - When the chmod call changes group permissions, then the change is applied to the mask if the object has an ACL.
  - This ensures users using non ACL-aware tools don't grant more access than they intended to users or groups with existing ACL entries.

POSIX ACL evaluation

- A POSIX process has an associated effective user id (euid), effective primary group id (egid), and a list of additional groups (gid's).
- When checking the requested access (rwx) against an object with a POSIX ACL, the order of evaluation is as follows :
  - uid matches are made first (starting with the owner uid). If any uid entries in the ACL match, this entry is used for access.
  - Search for any matching gid entries, if the requested access is granted for any gid associated with the process then allow access.
  - If no other entry matches, use the "other" entry for access.

"Overdesigned, underused and added to NFSv4" - Win32 ACLs

- Win32 ACLs are (IMHO) a mess.
  - Beautifully designed from a computing science point of view, they are so complex to use that almost NO Windows administrator understands them.
  - In addition, so few Win32 programmers understand them that in practice most applications also ignore ACLs.
  - Order dependent, moving the entries within an ACL can completely change the access decisions granted by that ACL.
- Win32 ACLs (like most things in Win32) are a moving target. Many changes introduced in Windows 2000.

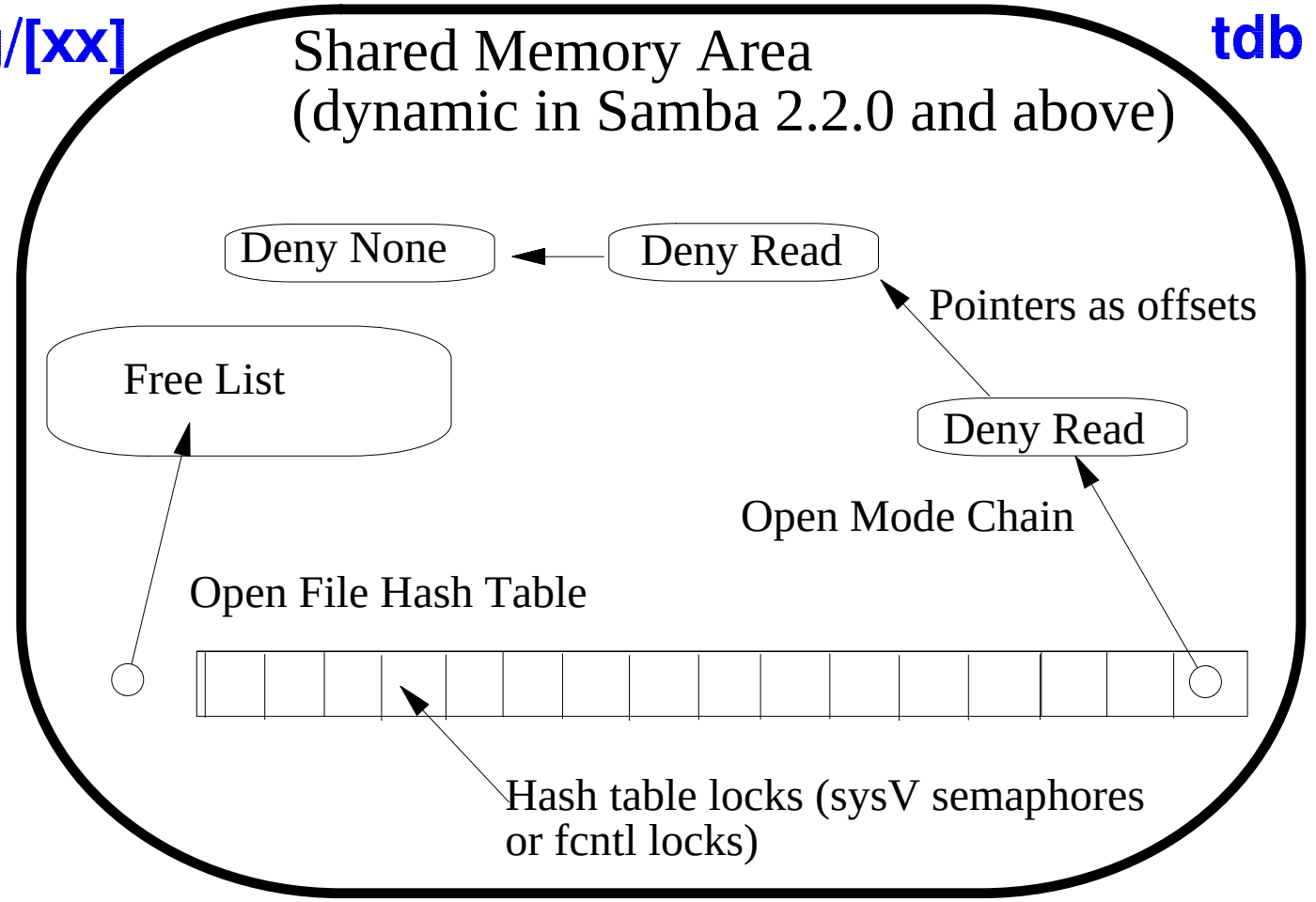
Deny mode semantics in POSIX

- POSIX has no "deny modes". Samba layers these over ordinary POSIX open calls [smbd/open.c].
  - POSIX apps do not interact with DENY modes.
  - Reason - what happens if someone opens /etc/passwd with DENY_ALL ?
  - DENY mode semantics are not logical - adding this to POSIX is not good design.
- Samba implements a fast, smbd to smbd mechanism to convey deny modes between user processes.
  - No centralized deny mode daemon needed.

Samba shared memory Deny mode database

locking/[xx] in 2.0.x

tdb in 2.2.0



Creating Oplocks in POSIX

- Allowing Oplocks on top of POSIX breaks consistent view of filesystem (and Samba philosophy) [smbd/oplocks.c]
  - However, too useful not to implement. Needed for SMB speed.
- Deny mode database holds all shared info about open file state. Oplock records added to this data.
- Blocking IPC mechanism between smbds needed that would integrate into select()/poll().
- UDP messages on loopback interface chosen.

Oplock communications

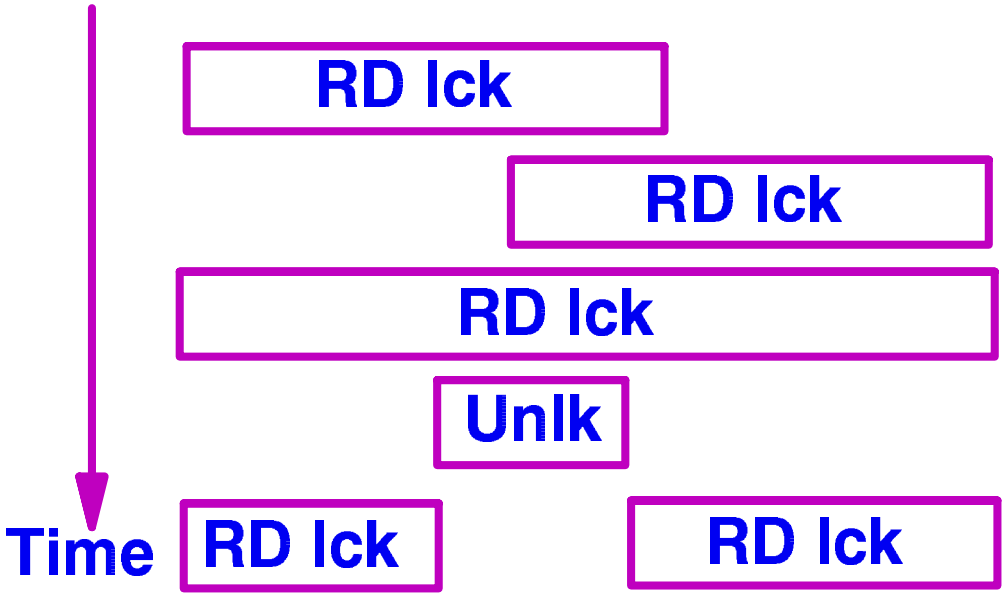
- On break request, smbd locks db, finds holder of oplock, sends break request via UDP port, releases db lock then blocks awaiting reply).
  - Code in [smbd/open.c] and [smd/oplock.c] - request_oplock_break() function.
- Receiver smbd gives priority to incoming UDP messages in select(), recurses into secondary smbd processing loop [smbd/oplock.c].
  - "Dangerous" messages that may cause an oplock break from the receiving smbd are queued at this time.
- On exit from recursed state, queued messages are given priority [smbd/process.c] - receive_message_or_smb().

The swamp - mapping Win32 byte range locks to POSIX

- Win32 byte range locks seem to be easy to map into POSIX.
  - Approach chosen in all Samba versions 2.0.x and before.
  - Depends upon locking conflicts being handled at client redirector.
  - Not possible to give exact Windows semantics.
- Samba 2.2.x and 3.0 have correct Win32 semantics.
  - "Correct" here means 'what NT does'. Has little relation to Win32 documentation or the spec.

POSIX locks - the exact semantics

- Lock ranges can be merged/split.
- Lock ranges can be upgraded/downgraded.
- 32/64 bit signed, not unsigned ranges.



Kernel view of locks

POSIX lock semantics (continued).

- Killer issue : POSIX locks are **per process**, not per file descriptor.
- Eg:

```
int fd1 = open("/tmp/bibble", O_RDWR);
fcntl(fd1, F_SETLK, &lock_struct);
fd2 = dup(fd1);
close(fd2);
```

SURPRISE ! The lock you thought you had on fd1 is now gone !

In anyones wildest dreams this is not desirable behavior.

POSIX lock semantics (continued).

- Samba 2.0.x solution to this problem was to reference count all opens on a file onto a single fd, open read/write (if possible).
  - Conserves fd usage.
  - Samba checks prohibited security overrides.
- Disadvantages are :
  - Multiple opens under different uids - need to use fork() as a procedure call to check return.
  - smbd is lying to operating system about access mode.
- 2.2.0/3.0 solution - store pending closes in a tdb.
  - Allows multiple opens to obey Samba philosophy.

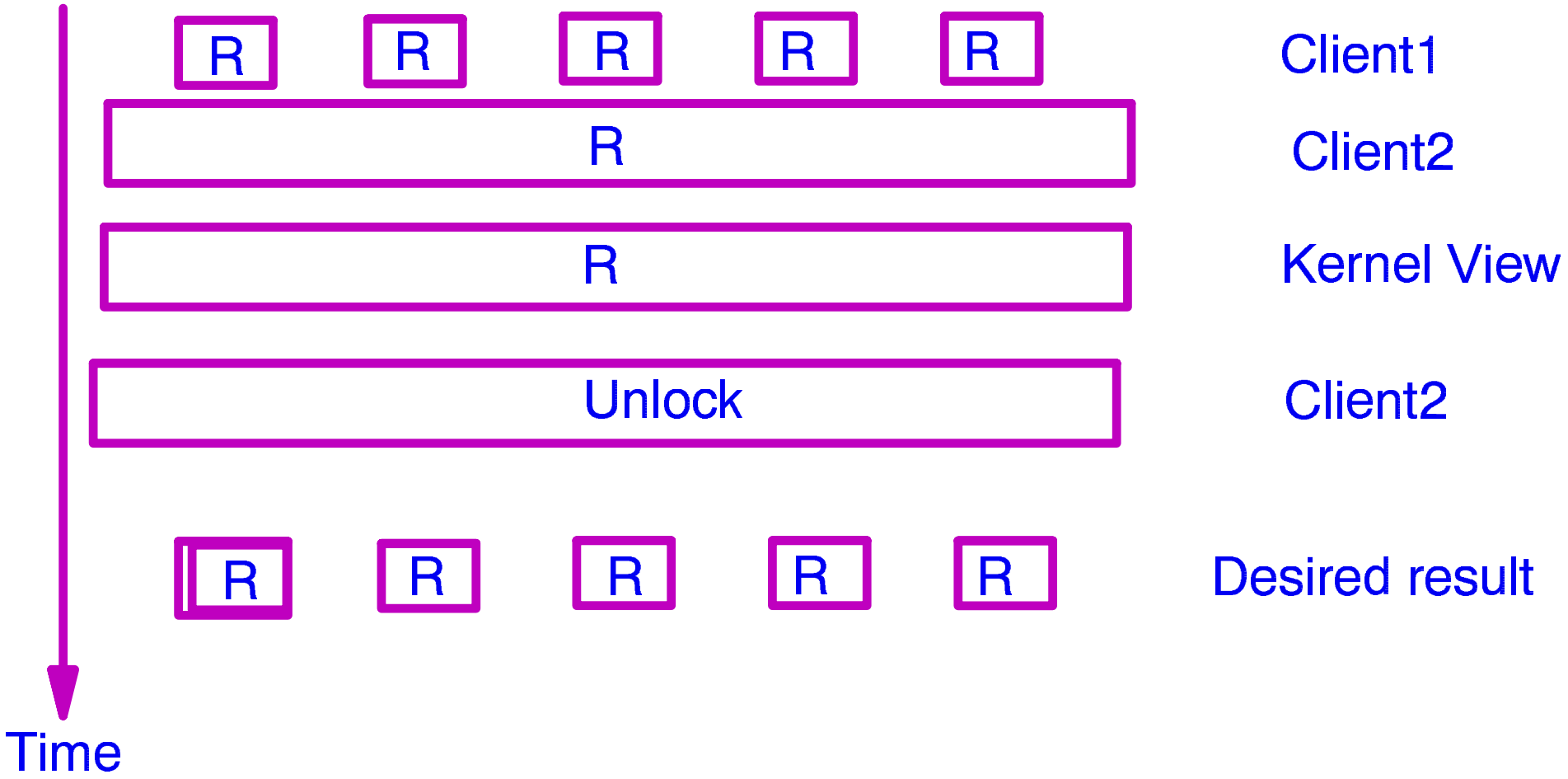
"Welcome to Fantasy Island" : The Win32 lock spec.

- Win32 locks as described in Win32 docs are not what is implemented in Windows NT.
  - Locks can be downgraded by overlaying read locks onto write locks and then doing one unlock.
  - Compatible locks can be stacked on top of each other and are then reference counted.
- The only way Samba can implement this is with a locking database.
  - This tdb database [locking/brlock.c] implements full 64 bit Win32 lock semantics, indexed by dev/inode pairs.
  - Any locks passed by this are (optionally) passed to a POSIX lock mapping layer [locking/posix.c].

Mapping Win32 locks to POSIX

- POSIX lock layer attempts to map given 64 bit unsigned lock onto signed (64 or 32, depending on filesystem) bit POSIX lock.
  - If no POSIX mapping possible - discard the request (return True - POSIX app can't get to this range anyway).
- Locks that pass are then stored in a second, lower level tdb that contains full record of all existent POSIX locks on a dev/inode pair.
  - This is needed as POSIX kernel will lose information when locks are overlapped.

Mapping Win32 locks onto POSIX (continued).



ChangeNotify and timed locks

- ChangeNotify is a problem as it is resource intensive.
  - Similar to FAM on IRIX ((kernel interface)- this is now available on Linux.
  - For portability reasons, Samba currently does a periodic scan, with no depth.
  - Produces a hash of the directory contents and checks this in the idle loop [smbd/nttrans.c].
- Timed locks are implemented by all lock requests being instantaneously checked with the request packet being queued until a check succeeds in the idle loop (or timeout) [smbd/blocking.c].

Samba DCE/RPC subsystem: incoming

- Pipe opens are done on a IPC$ share, smbd redirects into pipe handling code [smbd/pipes.c].
- All writes onto pipe handle are buffered into a continuously growing (length limited) memory buffer [rpc_server/srv_pipe_hnd.c].
  - On an authenticated RPC bind (NTLM handshake), the user credentials are stored with the pipe [rpc_server/srv_pipe.c].
  - As a PDU's worth of data is received, the header is processed, stripped off (all sign & seal processing is done here) and the incoming PDU data is appended.
  - When the complete RPC is received then the pipe/function specific processing is invoked.

Samba DCE/RPC subsystem: outgoing

- After successful processing of the RPC request the outgoing data stream is marshalled into an auto-growing buffer via [rpc_parse/parse_XXX] calls.
- When the client does a read on the RPC pipe the outgoing data is split into PDU sized chunks [rpc_server/srv_pipe_hnd.c] and returned as the read data.
- Additional pipes (eg. MS-DFS pipe) can be added into pipes tables in [rpc_parse/parse_rpc.c] - uuid, and [rpc_server/srv_pipe.c] - pipe function table.

Resources

- Main Samba Web site :
  - http://samba.org
- Newsgroup :
  - news:comp.protocols.smb
- Samba discussion list :
  - email: samba@samba.org
- Samba development list :
  - email: samba-technical@samba.org